# Эссе по теме:

# PHP UP | Урок №7 | Часть #1 | Пакетный менеджер Composer

Теги в git.

Представьте, что у вас есть ветка разработки в репозитории, в ней содержится большое количество коммитов. Это вполне типичная ситуация, когда нужно узнать в какой версии какие изменения содержатся. Можно пойти в историю ветки в репозитории и прочитать эти изменения, однако зачастую далеко не все члены команды хорошо разбираются в коде проекта. Для решения подобных ситуаций вводят такое понятие, как версии проекта. Например, разработчики договариваются определенный коммит отметить, как версию 1.0. Для каждой версии в какой-то специальной системе можно создать текстовый документ с кратким описанием возможности этой версии. Смысл состоит в том, что метку будут получать только какие-то важные коммиты, в которых определенная работа считается завершенной. Такой подход позволяет однозначно сказать какая версия проекта сейчас работает на главном сервере. Каждый человек команды, менеджеры, программисты или рекламщики могут прочесть нужный документ с описанием версии и понимать какие функции и возможности реализованы. Когда версия проекта обновляется в документ описывается информация о новой версии, ее еще называют ‘change log’ (история изменений).

Для того, чтобы рассмотреть все версии, которые доступны в данном репозитории нужно запустить консоль и вызвать команду: $ git tag –list.

Также можно выполнить еще одну команду, которая нам даст все те же версии, но с указанием коммита на который они ссылаются. Используя кэш мы можем переключиться на нужную версию к примеру на версию 2.0.4 если нам нужно посмотреть там что-нибудь.

Обратная совместимость.

К примеру есть первая версия библиотеки. Вы используете ее в проекте. Вы использовали первую версию, но со временем вышла вторая версия и третья версия. Если вы можете взять вторую версию или третью версию и подставить ее в свой проект, при этом ваш проект будет работать корректно, тогда можно сказать, что обновление версий не нарушают обратную совместимость. Если же вы обновляете библиотеку с более низкой на более высокую версию, и при этом ваш код перестает работать, тогда говорят, что обновление ломает обратную совместимость или нарушают обратную совместимость.

Один из самых популярных подходов для сканирования он называется SemVer, сокращенно от семантическое версионирование. Предположим, что в нашем проекте мы используем зависимость какую-то библиотеку версии 3.2.1.1.

Первая цифра (3) - это мажорная версия, как правило, она отличает какие-то кардинальные изменения в программе. Когда программа приобретает новую мажорную версию. Это значит, что она больше не обратно совместима со своей предыдущей версией, то есть если вы попытаетесь обновить библиотеку до четвертой версии, практически гарантированно, что код перестанет работать. Поэтому, это самая важная цифра и на нее нужно обращать внимание в первую очередь.

Следующая цифра (2) - это минорная версия, когда библиотека получают новую минорную версию это значит, что в ней содержится изменения, которые не затрагивают обратную совместимость. Практически всегда это значит, что вы можете спокойно обновлять библиотеку в пределах минорной версии, при этом ваш код, который зависит от этой библиотеки будет продолжать работать. Третья цифра (1) - это патч версия. В патчах обычно содержатся изменения, которые отвечают за исправление ошибок в программе. Количество патчей, как правило, больше чем количество минорных и мажорных версий.

Публикация пакетов на packagist.org.

Достаточно часто встречается ситуация, когда вы разработали какое-то универсальное решение определенной проблемы. Вы внедрили это решение в текущий проект, и она себя хорошо зарекомендовала. В следующем проекте вы снова хотите использовать этот класс. Самым простым способом было бы просто скопировать нужные файлы в папку проекта и работать с ними, но на практике это не удобно, особенно если ваше решение будет использовать другой разработчик. В этом случае логично поместить свою наработку в собственный composer пакет. При этом нужно учесть, что содержащаяся внутри пакета логика должна работать в определенной предметной области. Представим, что мы хотим использовать наш какой-либо класс, как composer- пакет. Для этого нужно создать и оформить пакет локально. При помощи программы composer опубликовать код в репозитории, создать и настроить новый пакет на сайте packagist.org - это хранилище пакетов composer. После этого можно использовать пакет, если вы сделаете пакет публичным, то любой желающий сможет использовать пакет в своем проекте. Также есть возможность сделать ваш пакет приватным.